AD103045

Semiannual Technical Summary

DTIC
AUG 18 1981
H

Distributed Sensor Networks

30 September 1980

Prepared for the Defense Advanced Research Projects Agency
under Electronic Systems Division Contract F19628-80-C-0002 by

Lincoln Laboratory

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

LEXINGTON, MASSACHUSETTS

Approved for public release; distribution unlimited.

The work reported in this document was performed at Lincoln Laboratory, a center for research operated by Massachusetts Institute of Technology. This work was sponsored by the Defense Advanced Research Projects Agency under Air Force Contract F19628-80-C-0002 (ARPA Order 3345).

This report may be reproduced to satisfy needs of U.S. Government agencies.

The views and conclusions contained in this document are those of the contractor and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the United States Government.

This technical report has been reviewed and is approved for publication.

FOR THE COMMANDER

Raymond L. Loiselle

Raymond L. Loiselle, Lt. Col., USAF
Chief, ESD Lincoln Laboratory Project Office

Non-Lincoln Recipients

PLEASE DO NOT RETURN

Permission is given to destroy this document when it is no longer needed.

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

LINCOLN LABORATORY

DISTRIBUTED SENSOR NETWORKS

SEMIANNUAL TECHNICAL SUMMARY REPORT
TO THE
DEFENSE ADVANCED RESEARCH PROJECTS AGENCY

1 APRIL – 30 SEPTEMBER 1980

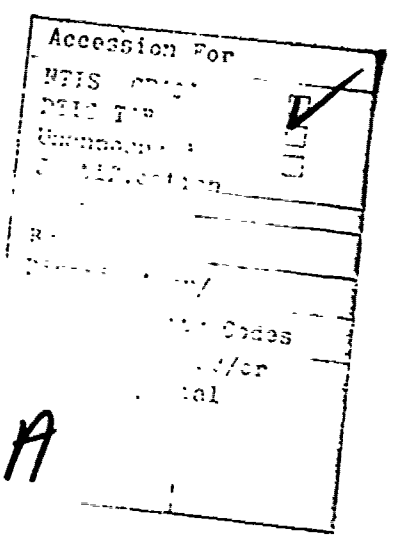
ISSUED 12 JUNE 1981

Approved for public release; distribution unlimited.

LEXINGTON MASSACHUSETTS

ABSTRACT

This Semiannual Technical Summary reports work in the Distributed Sensor Networks program for the period 1 April through 30 September 1980. Progress related to development and deployment of test-bed hardware and software, including deployment of three test-bed nodes, is described. A complete algorithm chain from raw data to aircraft locations, employing two acoustic arrays, has been developed and demonstrated experimentally using data collected from test-bed nodes. A strawman design for a new multiple microprocessor test-bed node computer is presented. Also described is progress in the design and development of a real-time network kernel for the DSN test bed in general, and the new processor in particular.



CONTENTS

DISTRIBUTED SENSOR NETWORKS

I. INTRODUCTION AND SUMMARY

This Semiannual Technical Summary (SATS) for the Distributed Sensor Networks (DSN) program reports research results for the period 1 April through 30 September 1980. The DSN program is aimed at developing and extending target surveillance and tracking technology in systems that employ multiple spatially distributed sensors and processing resources. Such a DSN would be comprised of sensors, data bases, and processors distributed throughout an area and interconnected by an appropriate digital-data communication system. It would serve users who are also distributed within the area and serviced by the same communication system. Of particular interest is the case when individual sensors cannot view the entire surveillance area and when they can individually generate only limited information about targets in their field of view. The working hypothesis of the DSN program is that, through suitable netting and distributed processing, the information from many such sensors can be combined to yield effective and serviceable surveillance systems. Surveillance and tracking of low-flying aircraft, including cruise missiles, using sensors that individually have limited capabilities and limited fields of view, has been selected to develop and evaluate DSN concepts in the light of a specific system problem. The research plan is to investigate these concepts and to develop a DSN test bed which will make use of multiple small acoustic arrays to detect and track low-flying aircraft.

Progress in the development and deployment of the test bed at Lincoln Laboratory is reported in Sec. II. This effort includes modifications to node designs, deployment of additional nodes, work on providing wire communication between nodes and a monitor and control computer, preparations for mobile nodes, further development of the real-time kernel for the nodes, and work on developing a real-time signal-processing capability for the nodes.

The first data-acquisition node located on the Laboratory's Building L has been retrofitted to improve performance and ease of use. The physical array was reconfigured using individual tripod mounts for the microphones. The system was modified to improve its ability to resist outdoor conditions for long periods of time and to allow complete operation of the system from inside. Calibration, audio-communication, and audio-recording features were added to improve ease of operation, improve experimental control, and provide better experiment documentation. A second node, identical to the retrofitted L node, was constructed and deployed with the array being on the roof of Building J in the main Laboratory complex. A third array, with an experimental rigid but easily configured frame holding microphones, has been deployed on the roof of a hangar at the Lincoln Flight Facility. Communication between the Flight Facility and the main Laboratory is provided by 9600-baud short-haul modems with unloaded lines.

A new version of the real-time kernel (DAK) has been developed. It provides for limited access to all the memory in a node and will be the basis for all real-time processing in the nodes for the next several months. This real-time processing includes data collection and recording on digital tape, for which drivers and other user software must still be converted to the new kernel, and real-time signal processing which will involve a Floating Point Systems array processor as well as the PDP-11/34 in each node. Progress on the development of real-time software to control and make use of the array processor as well as actual array-processor software is also described in Sec. II.

Section III reports on data-collection activities involving known and controlled aircraft and upon the successful development and experimental demonstration of a complete data-processing chain from raw data through to two-site acoustic locations of aircraft. Also described are modifications and improvements to the Acoustic Data Analysis Program used for algorithm development and experimental data processing.

Section IV reports on development of a REal-time NEtwork kernel (RENE) which is to be a successor to DAK, and also on a strawman hardware design for multi-microcomputer node architecture which could be used to enhance test-bed node capabilities and would be a step toward smaller, low-power nodes in the future. The software kernel is to be implemented for the existing PDP-11 node computers as well as for any multi-microcomputer system which might be developed. It is a real-time system being specifically designed for use in a distributed multicomputer, multi-node environment. The strawman hardware design is modular and can be used to realize very powerful DSN nodes. It uses Motorola M68000 chips as the main processing element, and incorporates many features related to reliability, maintainability, and ease of debugging hardware. In order to decide how next to proceed, we are presently at a stage where DSN requirements and the strawman design are to be reviewed in the light of commercially available microcomputers.

Finally, miscellaneous activities including interactions with other research groups, relocation of our physical facility within the Laboratory, and conversion to Version 7 UNIX are summarized in Sec. V.

II. TEST-BED DEVELOPMENT

Progress in the development and deployment of hardware and software for the DSN test bed is reported in Secs. A through C below. Section A addresses hardware and deployment; Sec. B deals with the real-time software kernel to support data acquisition and other real-time processing; and Sec. C reports progress on the real-time signal-processing software for the deployed nodes.

A. NODE HARDWARE AND DEPLOYMENT

The hardware configuration of the Data Acquisition System described in our previous SATS[1] has been significantly improved. In addition, two more microphone arrays have been constructed and installed on the roofs of two other Laboratory buildings. One of the new sites is within the main Laboratory complex, and the other is remotely located at the Lincoln Flight Facility. We now have a total of three DSN arrays in place for data-collection experiments. The two sites at the main Laboratory complex are completely operational. The third is awaiting delivery of a tape system and final checkout and deployment of the electronic racks at the Flight Facility. Two of the nodes contain FPS 120B array processors for future real-time signal processing. The array processor for the third node is temporarily attached to our PDP-11/70 research support computer for software development purposes. Options for data communications between remote nodes and the PDP-11/70 which also serves as our monitor and control computer have been investigated, modems have been procured, and an initial 4-wire line with 9.6 kbits modem has been installed between the existing remote site and the main Laboratory. Plans for mobile DSN nodes have been formulated, and are in the process of being reviewed and finalized. Additional details are described in the following paragraphs.

The square and rigid array that was initially installed on the roof of the Laboratory's Building L has been reconfigured into three concentric equilateral triangles with a microphone placed at each of the vertices. Figure II-1 shows the configuration of the new array. The triangle base dimensions are approximately 6, 2, and 0.75 m. This new configuration was adopted to improve the spatial resolution and aliasing properties of the array over the 20- to 180-Hz band. The larger aperture gives better resolution at the lower frequencies, while the small spacings near the center avoid aliasing at the higher frequencies.

Separate supports are used for each microphone element of the new array. The metal tripods chosen as the element supports are easy to erect and align. The use of separate supports for each element allows easy reconfiguration as long as the array is deployed on a reasonably flat surface. Figure II-2 shows the physical construction of the new tripod array. There is now one such array located on the roof of Building L where the original data-acquisition array was located and a second on the roof of Building J, also in the main Laboratory complex. The third array on the roof of a hangar at the Lincoln Flight Facility has a different construction which is described later in this section.

A new preamplifier protection circuit and power distribution system has been designed. The originally deployed power supply consisted of non-rechargeable mercury batteries with individual preamplifier protection relays for each channel. The power supply, including switches, relays, and reset buttons, was located at the array, several hundred feet from the main electronics and operator's position. In the new design, all controls and components – other than some connectors and wires – have now been moved inside to the operator's position.

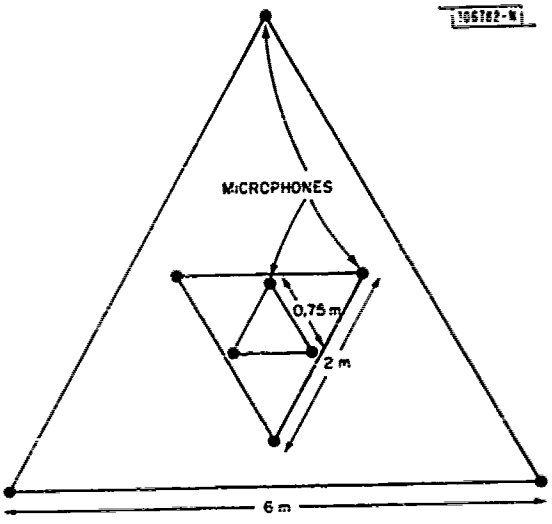


Fig. II-1. Tri-delta microphone array configuration.

Fig. II-2. Deployed tri-delta array using separate tripod mounts for each microphone and its preamplifier.

The new preamplifier power-supply design contains rechargeable batteries and an integral recharging system. The batteries consist of maintenance-free, sealed, lead-acid cells which are capable of maintaining a charge that is adequate to power an array for 4 to 6 weeks. The recharge system built into these cells incorporates recombination of gases with a starved electrolyte system. Therefore, the batteries can be enclosed within a sealed container.

A second change in the power distribution system design involves the use of new preamplifier protection circuits to replace the relays used in the initial design. The new circuit is entirely solid state with an adjustable current-limiting trip point to provide maximum protection of the microphone preamplifiers.

A multipair shielded cable which extends from the operator's position to the outside array is used to provide power to all channels. It is comprised of 19 individually twisted pairs, surrounded by a common shield. With the power supply and all controls now inside at the operator's position, experiments can be conducted more conveniently and safely, independent of weather conditions.

Providing remote power to the array necessitated the construction of a simple preamplifier/power-supply interface (PPI) which is to be located with the microphone array. The PPI, shown in Fig. II-3, couples the power cable, preamplifier cables, and individual channel signal lines at the array. Power is distributed from the PPI to each preamplifier (located physically at each microphone). The output of the preamplifiers is connected within the PPI to the proper signal cable lines which terminate at the A/D hardware located inside at the operator's position.

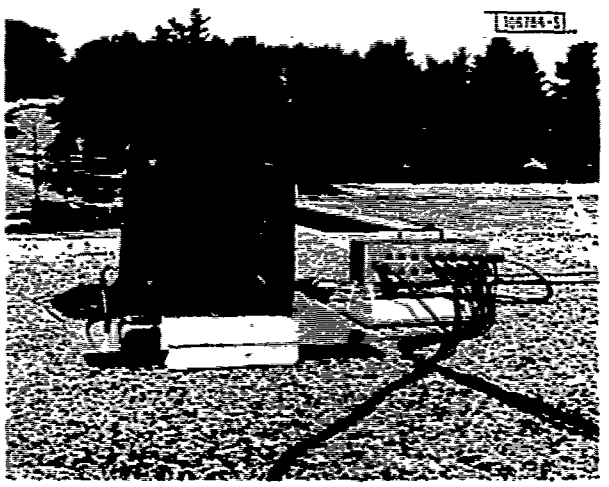


Fig. II-3. Preamplifier/power-supply interface units and calibration loudspeaker to be located outside at microphone arrays.

The existing three nodes contain temporary prototype versions of the new power-supply design. The location of the prototype power supply in the node electronics racks is shown in Fig. II-4(a-b). Use of the prototype power supplies has expedited node deployment and has provided an operational test bed for the final design. In the future, additional nodes will be constructed using simple PC boards to form a more reliable and easier-to-monitor final version of the power-supply system. The present nodes will be retrofitted with the newer design.

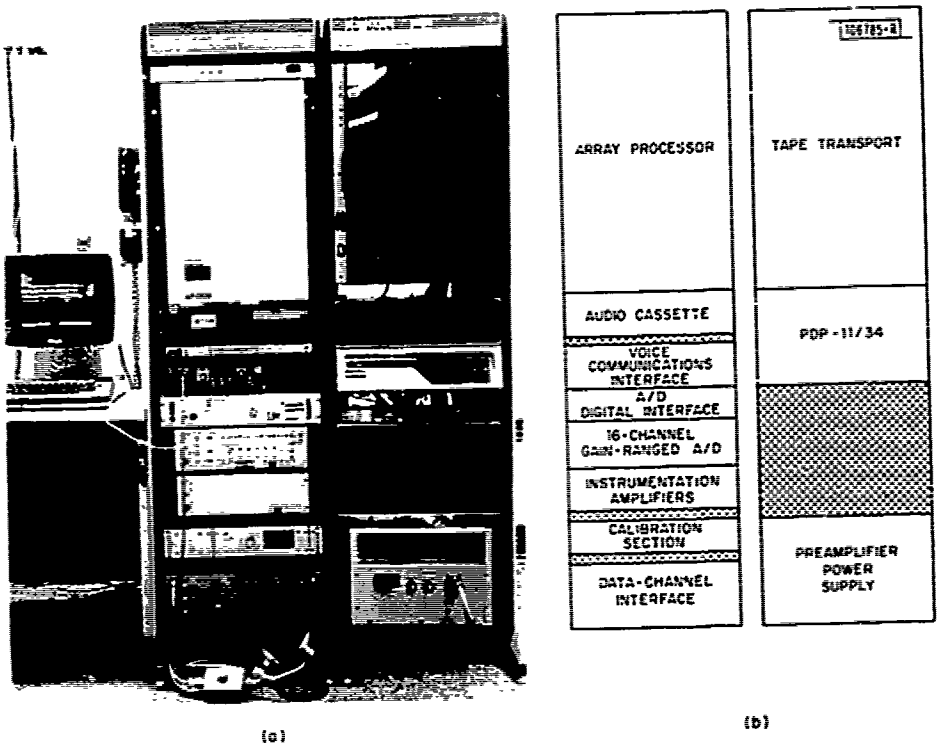


Fig. II-4. Node electronics and operator's position: (a) physical equipment; (b) location of subsystems within two racks of equipment.

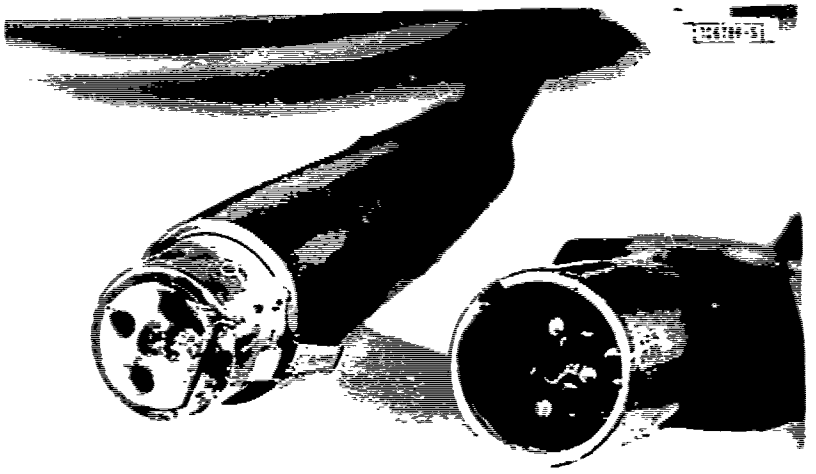


Fig. II-5. Deterioration of standard audio connectors after outside use for two-month period during summer.

As reported in the previous SATS,[1] measures were being taken to ensure that hardware exposed to the outside environment would be weatherproof. The most critical areas are the connectors where the signal lines, preamplifier lines, and power cable interface. In the initial system, general-purpose three-pin audio plugs and receptacles were used on a temporary basis due . very long lead times for connectors better adapted to outside use. A routine inspection of the Building L array a few weeks after deployment revealed oxidation of the audio connectors. Figure II-5 shows the condition of the preamplifier extension cable connectors after two months of operation during the summer. The condition of these connectors reconfirmed the need for the better plugs and receptacles which had already been ordered. The conversion process for the first array consisted of replacing all audio connectors with environmentally resistant, shielded, pressure-locking electrical connectors. Figure II-6 shows a microphone, preamplifier, and preamplifier cable with the new connector affixed. The connector pairs consist of 4-pin plugs, with gold-plated contacts sealed in a vulcanized rubber insert, mounted on the cables, and 4-pin bayonet lock receptacles with a similar insert, mounted on the preamplifier power interface. The two new arrays have been built using the new connectors and PPI.

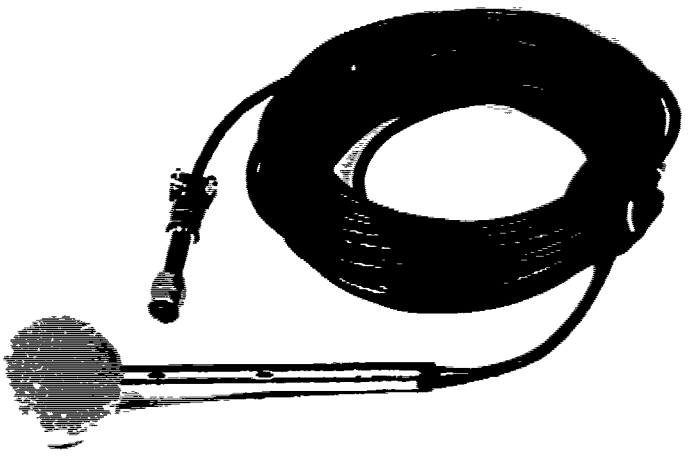


Fig. II-6. Microphone, preamplifier, and preamplifier cable with environmentally resistant, shielded, pressure-locking connector attached.

It was immediately apparent from the first few data-collection experiments that better mission documentation and operator control were necessary. Also, a method of quickly and easily confirming system integrity just prior to an experiment was required. All these functions were accomplished by adding operator support hardware to the nodes. The support hardware involves a cassette recorder, audio mixer, aircraft transceiver, tone marker, calibration meter, calibration oscillator, and loudspeaker. Figure II-4(a-b) shows the construction of the A/D hardware, support equipment, PDP-11/34 processor, array processor, tape drive, and power supply. The operator's position is just to the left of the equipment racks. The mixer, transceiver, and

tone marker are located in the voice communication interface. The calibration meter and oscillator are located in the calibration section. The data-channel interface consists of interconnections between the calibration section and the signal cables from the microphone preamplifiers which are remotely located at the array. As shown in Fig. II-7, the acoustic data flow is from the data-channel interface, through the calibrator section, to the instrumentation amplifiers which provide gain and anti-alias filtering through the gain-ranged analog-to-digital (A/D) converter and its digital interface, and to the PDP-11/34. With data-acquisition software running, the PDP-11/34 accumulates buffers of data and causes data to be stored on digital tape. When real-time signal-processing software becomes available, the acoustic data will flow from the PDP-11/34 into the array processor and the reduced data from the array processor will be returned to the PDP-11/34.

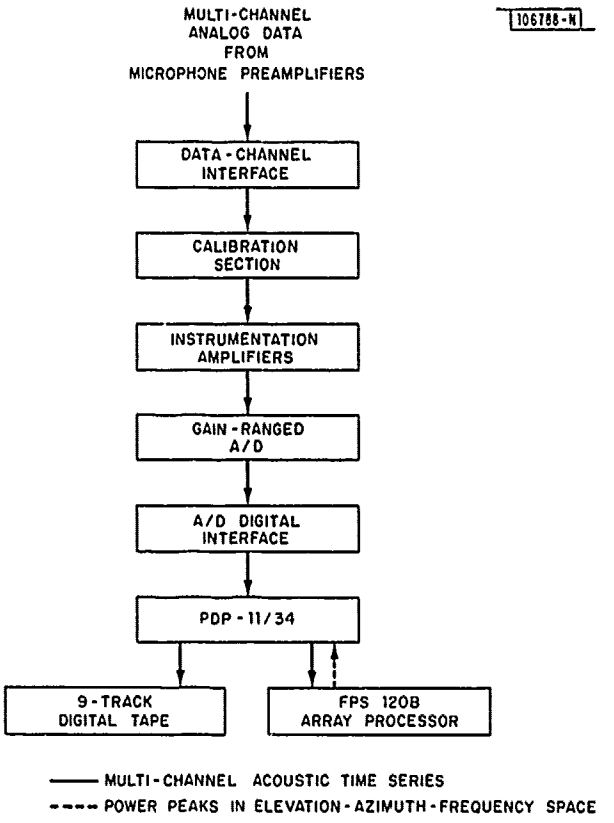


Fig. II-7. Primary data flow in a node.

Each operator can now communicate with the pilot, and with all other node operators using the aircraft transceivers. This provides essential communication for conducting controlled experiments, and also provides for an interim multi-node synchronization mechanism for starting data runs. The audio cassette recorder and audio mixer are used to make an audio tape history of all communications between the aircraft, the mission spotter located at the array, and all other DSN node operators. One recording channel of the cassette is used for this purpose. In

addition, a single audio channel of the microphone array is recorded on the other channel of the cassette recorder. This channel is easily reviewed and analyzed to aid in interpretation of the digital data collected during an experiment.

The calibration oscillator and gain calibration meter which are part of the equipment at the operator's position, and a loudspeaker which is installed at the array site as shown in Fig. II-3, are used to do a quick continuity and calibration check of all channels just prior to an experiment. The oscillator drives the loudspeaker with a specific calibration frequency. Each channel is individually monitored using the gain meter to determine if the correct signal levels are being received for the known signal level from the oscillator. Absolute calibration is occasionally done manually by visiting each microphone with a special absolute calibrator. Immediately after such an absolute calibration, the loudspeaker is driven with a known signal level and the received signal levels are recorded for subsequent quick calibration. The tone marker in the voice communication interface circuit is controlled by the operator. When the tone marker is enabled, it places a short tone burst on the audio tape and on a spare channel of the digital-data tape to indicate a significant event. This is used to mark times when controlled experimental aircraft pass checkpoints, and it is also used to manually provide synchronization information on data tapes.

The DSN now consists of three arrays located in the vicinity of the main Laboratory complex. In the near future, real-time processing of a portion of the data at the nodes is planned, with transmission of reduced measurements back to the PDP-11/70 where location and tracking will be done. This will be the first of our real-time experiments. For the next several months, all intercomputer communications will be via transmission over either the dial-up telephone system or leased (unloaded) 4-wire private-line metallic circuits. To this end, two pairs of short-haul modems and a pair of long-haul modems have been obtained. At the present time, we plan to connect the remote PDP-11/34 based nodes to the central PDP-11/70 via 9.6-kbaud circuits. The link between the Lincoln Flight Facility node to the PDP-11/70 is operational. The line between the Antenna Test Range and the PDP-11/70 is in the process of having the inductors removed so that it will be compatible with the short-haul modems.

We are now in the process of developing mobile DSN test-bed nodes which can be deployed easily. This will allow for experiments with various node configurations and for experiments at a variety of locations. Detailed specifications for a motor vehicle that will house all electronic equipment for a mobile DSN node have been completed. The proposed vehicle will be a relatively heavy-duty conventional truck and cab with a custom-designed body housing. The housing will accommodate three 6-ft equipment racks, a desk with computer control terminal, electronic test equipment, and operating personnel.

Primary AC power for the mobile-system electronic and electrical equipment will be supplied by public utility sources when available. Otherwise, a gasoline-driven 15-kW motor generator (MG) set will be used. The MG set will be installed on a shock-mounted platform to the rear of the body housing on-board the vehicle.

The nature of DSN tests requires that ambient background-noise levels at the low end of the audio band be suppressed to enhance detection and tracking of targets. In the mobile system, the motor generator is a potential source of unwanted background noise. To suppress this noise, the MG will be housed in a sound-attenuating enclosure and the sensor array will be positioned approximately 300 m from the vehicle. If experiments indicate that generator sound levels are still too large, we will add batteries and an inverter to facilitate quiet operation for short periods.

To simplify the positioning of the individual sensors of the array in the mobile systems, a lightweight triangular framework will be used. The framework is an erector-set design, and can be assembled or disassembled in a matter of 20 to 30 min. The frame contains adjustable legs with swivel foot pads to allow leveling when set up on sloping or uneven surfaces. One of these frames has been constructed and installed on the roof of the Lincoln Flight Facility hangar as shown in Fig. II-8. The microphone stations are attached to the framework by "U" bolt clamps and, once correctly positioned, need not be readjusted even though the frame is disassembled and reassembled at a different location.

Fig. II-8. Deployed tri-delta erector-set array at Flight Facility.

B. SINGLE-PROCESSOR REAL-TIME KERNEL

For the last six months we have been running a Data Acquisition System (DAS) under Version 1 of the Data Acquisition Kernel (DAK). Both DAS and DAK have been described in a previous SATS.[2]

We now have under development a successor to DAK called the REal-time NEtwork kernel (RENE) as described in our last SATS.[1] To produce RENE we split DAK into two parts: an operating system independent part, called the Object Structured Discipline (OSD); and a part containing communication, scheduling, and memory-mapping primitives called RENE. In the last six months we have implemented the basic part of OSD, revised DAK to create DAK Version 2 (or DAK2) which uses OSD, and pursued separately the design of RENE proper. DAK2 will support all test-bed experiments during the next several months while RENE is being developed.

In revising DAK into DAK2 we have brought all documentation up to date, provided for access to more PDP-11 memory, and, by virtue of OSD, provided for easier possible conversion to

other computers. In the near future, DAK2 will also provide improved communication services between nodes and programs on the monitor and control PDP-11/70.

In the course of updating the documentation many minor improvements were implemented and, in addition, as a consequence of OSD development, the part of DAK2 written in the C programming language should now be able to run on other computers, such as the Motorola MC68000. This portability of software is important to us as we start development of a communication module and a new multiple microprocessor test-bed node based upon MC68000's. We have in fact imported an MC68000 C compiler from Steve Ward's group on the M.I.T. campus, and successfully compiled DAK2, but have not tested the runability of any compiled code.

DAK2 supports use of the full 248K bytes of PDP-11/34 memory in the simplest possible way. The first 56K is directly addressable by the processor, while the rest may only be accessed by some I/O devices and by a single kernel subroutine that copies a block of memory from any place to any other place.

DAK2 is currently up and running, but conversion of the A/D and magnetic-tape drivers and DAS program is not yet completed. The DAK2 communication design, reviewed immediately below, is completed and implementation is about to begin.

Communications between the PDP-11/34 at any node and our PDP-11/70 will be accomplished by a single ASCII full-duplex connection with XON/XOFF flow-control protocol, using standard terminal interface hardware and the natural ability in DAK to time-share the terminal port between several processes so that several independent DAK2 processes may send data back to the PDP-11/70. Binary data will be transmitted by encoding them into ASCII lines. This communications scheme, which is an enhancement of the system reported in a previous SATS,[2] will support test-bed applications for the next several months without requiring a major implementation effort.

The OSD mentioned previously consists of conventions and routines for passing large objects between subroutines, controlling errors and process exceptions, allocating memory, and performing basic input/output. In UNIX such a system exists at the program level, based on the UNIX shell language and the file system, but not at the subroutine level, where a real-time system needs it. A version of OSD existed within DAK Version 1. It now has been revised and made into an independent system capable of running with any operating system.

In addition to basic OSD, which is now implemented and running with DAK2 and UNIX, a number of higher-level OSD objects have been designed for use by RENE and applications programs. The higher-level objects, for which we have preliminary designs, include memory pools, a memory allocation system, byte strings used to store and access character strings, output streams for outputting text and graphics, input streams to provide a uniform syntax for data and command entry, and pointer lists for possible artificial intelligence applications. We plan to implement these and the other new OSD objects over the next few months as a basis for the implementation of RENE. Since this is being done in the OSD context, these capabilities will be available for use by DAK2 programs as well as RENE and should be easily portable to other computers for which a C compiler is available.

C. ARRAY-PROCESSOR SOFTWARE

In May 1980, we accepted delivery of three Floating Point Systems (FPS) AP-120B array processors. Each of these computers, capable of 12 million floating-point operations per second, provides the computational power for real-time signal processing at a DSN node. Two of the

processors were installed directly in DSN nodes, and the third was attached to our PDP-11/70 system running UNIX for software development. In addition to the hardware installation, several thousand lines of software for the array processors were installed on the PDP-11/70 UNIX system: a cross assembler, linker, debugger, diagnostic programs, etc. Much of this software was written in Fortran, but not Fortran 77 which is now standard for Version 7 UNIX. Installation thus also required some conversion. In addition, since the DSN nodes contain no provision for mass storage devices and will be remote from our central computing facility, downloadable diagnostics had to be developed by the manufacturer and tested at our site.

Three software packages were designed and are under development to make use of the array processors. The first package is the Signal Processing System (SPS) which runs on a PDP-11/70 with an attached array processor and is a software development package. It will also aid in non-real-time data analysis. The second, the analysis server, uses the DAK operating system and is targeted for real-time use in nodes. It controls data flow and real-time signal processing in the DSN node. Lastly, the array-processor server will provide a DAK system interface to the array processor in the test-bed nodes. The current state of these systems is described in more detail below.

The SPS is a package for developing real-time DSN signal-processing software. SPS operates on a dual-processor system consisting of a PDP-11/70 minicomputer, running the UNIX operating system, and a Floating Point Systems AP-120B array processor.

SPS has two major applications in the DSN project. First, SPS will speed up offline processing. Currently, the Acoustical Data Analysis Program (ADAP) (see Sec. III) runs can require several hours of PDP-11 time. These can be shortened to several minutes (near real time) using an array processor. The plan is to eventually interface SPS-developed array-processor software to ADAP to achieve that goal. Second, SPS permits us to experiment with and develop new algorithms for real-time data reduction. The FPS code so developed and debugged can be directly transferred to the DSN node for real-time uses. Thus, SPS provides an environment for development of real-time node software without the need to operate in that real-time environment from the start.

SPS currently provides for low-resolution frequency-wave-number signal analysis. It is in a state of rapid growth with high-resolution analysis to be provided as soon as matrix inversion software is developed for complex Hermitian matrices. The following small sample of commands displays the nature of SPS capabilities.

Command – csdm: Forms cross spectral density matrices from channel data

INPUT	Data from channels
	List of frequencies at which to form csdm's
OUTPUT	Average power
	Cross spectral density matrices

Command – hform: Complex Hermitian form calculator

INPUT	Complex vector
	Complex matrix
OUTPUT	Answer = [transpose (vector) * matrix * conjugate (vector)]

Command – peak: Power and peak calculator

INPUT	Array of frequencies at which to calculate power
	Array of wave numbers at which to calculate power
	Array of azimuths at which to calculate power
	Array of cross spectral density matrices
OUTPUT	Array of power peaks

Of the above, "hform" is a relatively low-level primitive command, whereas the other two are examples of substantially higher-level commands. The higher-level commands are composed of sequences of more primitive commands which also can be used directly.

The analysis server is a DAK2 software device which will reside in a DSN node and control calculation of target parameters from raw data. It will manage the real-time data flow and signal processing within the node. It is planned to be operational by January 1981, at which time it will implement initial algorithms which have been developed and demonstrated to work under good signal-to-noise conditions (see Sec. III). The preliminary design of the initial analysis server is now complete. Several hundred lines of C code, with documentation for the analysis server, have been written, deskchecked and compiled, but not debugged. Following is a more detailed exposition of the functions and operation of the analysis server and the real-time processing it controls.

The analysis server must perform three important activities. First, it manages the flow of data from the A-to-D server to the array-processor server. Second, it controls array-processor execution of DSN algorithms on these data. Lastly, the analysis server communicates with a DSN user process, accepting commands and returning power peaks corresponding to possible targets.

The analysis server will manage a dataflow of approximately 20,000 samples per second from the A-to-D server to the array-processor server. It acquires new data by sending empty buffers to the A-to-D server and receiving full buffers back on its return queue. Buffers are not, in general, physically moved in this process. Headers of data buffers received from the A-to-D server are checked for errors and the buffers are then sent, along with processing commands, to the array-processor server for analysis. After the array-processor server has accepted its commands and written the data into array-processor memory, the buffer is returned to the analysis server and then back to the A-to-D server, where the cycle begins again.

Following is a sketch of a typical processing sequence controlled by the analysis server. It is a two-phase operation. The first phase is to reformat data, take Fourier transforms, and generate power spectral density matrices. The second is to do wavenumber analysis and find power peaks. Data are received in multiplexed form in buffers holding a block of 512 samples from each of 9 channels. Eight such buffers constitute an analysis interval, 2 s of data. The software will accommodate changes in the number of microphones, sampling rate, block sizes, blocks per analysis interval, etc., but there will be restrictions imposed by available FPS and PDP-11/34 memory.

The first step of the first phase in the array processor is that each block must undergo conversion from A-to-D converter format (14 bits of mantissa, 2 bits of gain) to array-processor floating-point format (10-bit binary exponent, biased by 512, and 28-bit 2's-complement mantissa). Next, each channel of the block is fast Fourier transformed. From the transformed vectors, cross spectral density matrix accumulators are updated for each frequency listed in

array-processor memory. A full analysis interval of data (8 blocks) is processed in this way before going on to the second phase. Average single-channel spectra may be accumulated and used for resetting the frequency list to be used for the next analysis interval. The analysis server will have queued all the required blocks and commands on the array-processor queue by the beginning of the analysis interval.

The second phase of analysis takes the averaged cross spectral density matrices generated by the first phase as its input data. At each frequency of interest, a set of complex steering vectors will be computed and the power spectral density matrix will be inverted. The power is then estimated by a simple Hermitian form evaluation for each elevation and azimuth of interest. From the array of power values, indexed by elevation and azimuth, power peaks are found and saved for later return to the PDP-11 host. When all frequencies have been analyzed, the peaks are returned to the analysis server. A time-out protection mechanism ensures that the array processor will complete its calculations before the end of the analysis interval.

The analysis server must also communicate with a user process in the DSN node. This user may issue commands to open, close, start, and stop the analysis server. After receiving the open command, for example, the analysis server must open the A-to-D server and the array-processor server as well as a software clock to be used for time-out purposes. The start command causes the analysis server to send an analysis parameter object to the array-processor server which will initialize its memory with the analysis parameters; then buffer flow must be established, analysis begun, and peaks sent to an output stream.

The array-processor server, which is distinct from the analysis server, is the software which will provide a DAK interface to the array processor. It will handle commands to read, write, open, and close the array processor. It is the basic mechanism for executing commands on the array processor. It will allow users to declare functions to be executed on the array processor, initialize them, set parameters in them, load them, and execute them. The server will also manage memory in the array processor, provide bootstrapping service, and handle various exceptions and hardware interrupts.

The DAK2 array-processor server module currently consists of some 600 lines of C code and 200 lines of documentation. The code has been deskchecked and compiled, but not debugged.

REFERENCES

1. Semiannual Technical Summary, Distributed Sensor Networks, Lincoln Laboratory, M.I.T. (31 March 1980), DTIC AD-A091766.
2. Ibid. (30 September 1979), DDC AD-A086800/0.

III. ACOUSTIC EXPERIMENTS, ALGORITHMS, AND SOFTWARE

Data-collection activities and the development and experimental demonstration of a complete processing chain from raw data to 2-node acoustic locations are reported in Secs. A and B below. In Sec. C, additional details are presented of the alpha-beta filter which has been incorporated into the single-site azimuth tracking part of that processing chain. Section D reports additions to and modifications of the experimental signal-processing package which is used to aid in the development and testing of algorithms.

A. EXPERIMENTS PERFORMED

One important aspect of the DSN program is the collection of acoustic data that can be used to test new algorithms as they are developed. These data are collected in conjunction with the flight of an aircraft over a known course so that experimental and theoretical results can be compared. The data consist of microphone time-series data which are recorded at one or more nodes onto 9-track digital tapes. In addition, a 2-channel analog cassette tape is recorded at each site. These tapes, together with a record of the aircraft flight path, are then archived in our data library.

The 2-channel analog tape serves two purposes. One channel is used to record the sound at the array using a separate microphone. The other channel is used to record the conversation between the node operator, the rooftop spotter, the mission controller, and the pilot. These tapes are used to help analyze an experiment in several ways. First, the speech track is played back to ascertain when the pilot was over certain marks on the ground. Secondly, the comments of the spotters are used to aid in associating results with other air and surface traffic. Thirdly, the sound channel can be listened to, or played back through a real-time spectrum analyzer to determine what spectral components were present at any time in the environment.

The major experiments performed thus far are detailed in Table III-1. They consisted of three helicopter flights and one flight with a fixed-wing propeller-driven T28 aircraft. These

TABLE III-1
EXPERIMENTS CONDUCTED

Aircraft	Range (km)		Conditions		Array on Building
	Minimum	Maximum	Wind	Environment	
Small Bell Helicopter	0.5	4.0	Light	Moderate noise	L†
Small Bell Helicopter	0.5	3.8	Gusty	Noisy	L
T28 Propeller Aircraft	0.6	6.0	Light	Noisy	L, J
HU1 Helicopter	0.7	8.0	Gusty	Moderate noise	L, J

†With rectangular array; all others used tri-delta array.

experiments were conducted under differing environmental and wind-noise conditions, as noted in the table. All data except for the first experiment were recorded using the new tri-delta arrays, whose configuration is shown in Fig. II-1. Both the T28 aircraft and the UH1 helicopter were recorded at two nodes simultaneously.

In order to accurately locate ground marks, a member of the DSN team rode along as observer on the T28 and UH1 experiments. There was an operator for the node, as well as a spotter on each roof near each node, and an overall mission controller. As a result, these two experiments were very well controlled and documented, reflecting the lessons learned from the first two experiments. The only remaining problems are those of ensuring accurate time stamping of data and its coordination with aircraft location. These are being addressed by the planned incorporation of accurate time standards into the nodes, and the use of radar tracking information.

B. DEVELOPMENT OF ACOUSTIC TRACKING TECHNIQUES

During these past six months, we have successfully completed one major objective, namely the demonstration of algorithms to enable two nodes equipped with acoustic sensors to cooperatively locate low-flying aircraft. These algorithms involve a complete processing chain from raw-data inputs to aircraft locations as outputs. The algorithms have been experimentally demonstrated to work using data recordings of a T28 aircraft from two nodes. Locations and the true track are shown in Fig. III-1. The aircraft was tracked over some 2 km of its flight path with an accuracy of about 100 m except in the region where geometrical dilution of precision had a large effect.

While the results were excellent in proving some of the concepts employed, they should not be construed as being representative of the ultimate performance that can be achieved with acoustic sensors. The process of acoustic tracking involves a number of stages such as beamforming, frequency-target association, azimuth tracking, location, and location tracking. Until now, emphasis has been placed on finding a technique that would work at each stage rather than on optimizing the performance at each stage. It is anticipated that significant improvements in performance will be achievable once changes are made to the algorithms based upon the lessons learned so far. It is also worth noting that the general area of the Laboratory is a relatively high-acoustic-noise area.

The locations of the aircraft were obtained using an extension of the algorithm described in the prior SATS.[1] The algorithm produces locations given updates for an arbitrary number of azimuth tracks from an arbitrary number of nodes. But all locations are for pairs of sensor nodes. This algorithm builds lists of azimuth vs time for each of the azimuth tracks from each of the nodes. New azimuth-time points are then compared with each of the lists to ascertain where corresponding observations occur, and from these the locations are generated. To keep memory requirements within reasonable bounds, points are removed from the lists as soon as they are older than the longest time-difference-of-arrival between the nodes. It is expected that a version of this algorithm will be run in every node of a real-time system, and its output will be fed into a location-tracking filter.

A critical element for the successful determination of location is the generation of an azimuth-vs-time curve for a target. This requires several processing steps to be carried out every processing interval, which is typically 1 or 2 s. First, the time series from the microphones are analyzed to determine the directions-of-arrival at several different sound frequencies.

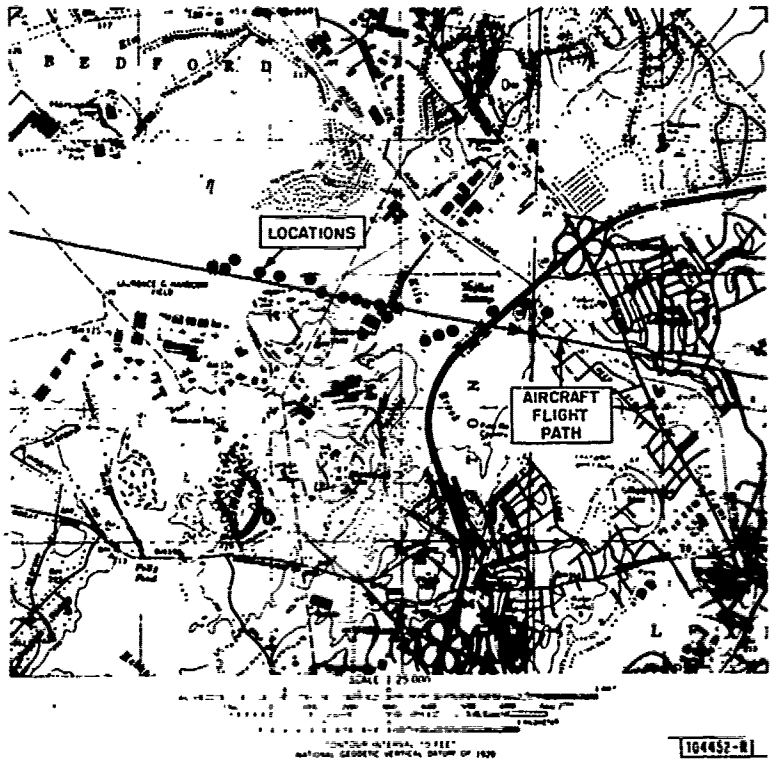


Fig. III-1. Acoustic locations of T28 aircraft produced by nodes on Buildings J and L. Straight line shows target track; solid dots show locations. Target heard at larger distances, but locations not calculated due to geometrical configuration.

Then, different frequency components are associated by direction-of-arrival into target azimuths, which are finally fed into target azimuth-tracking filters. These filters estimate the smoothed azimuth tracks which are used as input to the location process.

To obtain the direction-of-arrival of the frequency components, the MLM beamforming technique[1] was used. The analysis was done at eight frequencies with a resolution of 4 Hz. Since the arrays have good response from at least 20 to 200 Hz, and the targets had line spectra with widths less than 1 Hz in this band, our ability to detect the targets is not optimum. The best performance was obtained when the eight frequencies were selected to be the eight largest peaks in the average power spectrum. In general, all our initial algorithms tend to perform best when the target of interest is the dominant noise source. The limited number of frequencies and resolution were fixed upon in anticipation of conversion to real-time processing within individual nodes. We wanted a first-version algorithm which would work and which we could clearly see how to implement in the real-time nodes.

While the use of eight frequencies is suboptimal, investigation of the best means to choose them did disclose some interesting phenomena. By looking at analog recordings using a high-resolution real-time spectrum analyzer, we observed that the aircraft was rich in harmonics. We also observed that there were at least eight peaks visible in the 20- to 200-Hz spectra for most of the targets used for experimentation. However, we also observed that at any one time, several of these peaks could fade or be destructively interfered with by other noise sources.

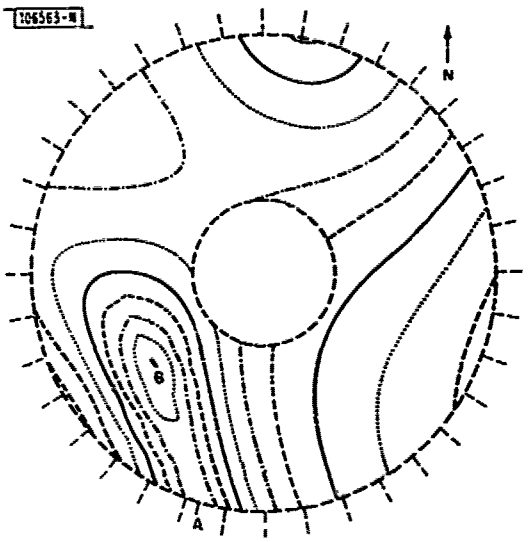


Fig. III-2. Contour plot of power vs wavenumber showing azimuth error which would result from examining power only on horizon. True peak at B, and false peak at A.

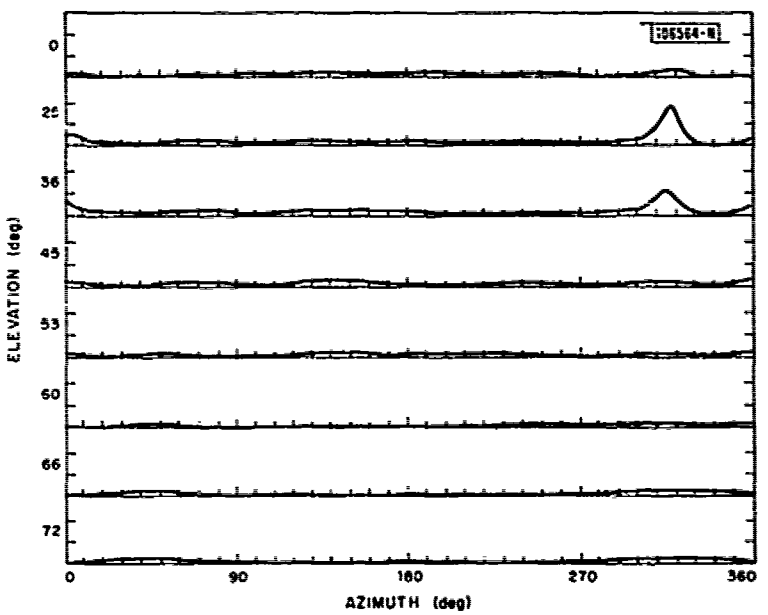


Fig. III-3. Typical measured power-vs-azimuth curves for T28 aircraft under good signal-to-noise conditions. Circles indicate maxima and diamonds indicate minima.

If all frequencies cannot be examined, then it is important to track the spectral peaks and not to rely on a small number of fixed frequencies for beamforming.

Optimum performance can probably be obtained by doing the MLM beamforming at all frequencies between the 20- and 200-Hz frequency limits with a 1-Hz or smaller frequency bin. This must be done for all azimuth and elevation values of interest, so the computational load required to do this is high. Also, it will exacerbate the problem of how to sort out the targets from the expected large number of peaks. However, the possible gains in performance are large enough that this will receive major research emphasis in the near future.

It was originally hoped that the computational load could be reduced by restricting the search for peaks of sound pressure to the horizon. Unfortunately, the peak response on the horizon can occur at a very different azimuth from the true peak. An example of this can be seen in Fig. III-2 which shows a contour plot of power as a function of wavenumber at a fixed frequency. In the figure, the response azimuth is represented by clockwise rotation and response elevation by distance from the center of the circle, with the outer ring representing the horizon. The true peak of response is marked B, whereas the component on the horizon is marked by A. We found that A moved erratically with time due to changes to the shape of the shoulders with interference. However, B moved steadily with target bearing. We concluded that it was necessary to compute the response at a sufficient number of azimuths and elevations to totally sample the wavenumber space.

The method used for finding the direction of each peak is a modified one-of-eight algorithm. The data are the power-vs-azimuth curves, computed for a number of elevations, at one of the selected frequencies, as shown in Fig. III-3. All the power-vs-azimuth curves are evaluated at the same set of azimuths. Each point has two neighbors at the same elevation angle, and three at each of the two neighbor elevations. The algorithm compares each point with its eight neighbors. If a point is higher than all its neighbors, then it is deemed to be a peak. This is simple to apply and produces good results. However, it does suffer from the problem of finding extraneous peaks when the sample intervals in azimuth and elevation differ significantly. An example of this is shown schematically in Fig. III-4 where one peak would be mistakenly reported as three. This problem was overcome by extending the comparison to enough circumferential neighbors to equalize the radial and circumferential wavenumber extent for comparison. This works in all but a few cases, which occurred infrequently enough to not affect the final results.

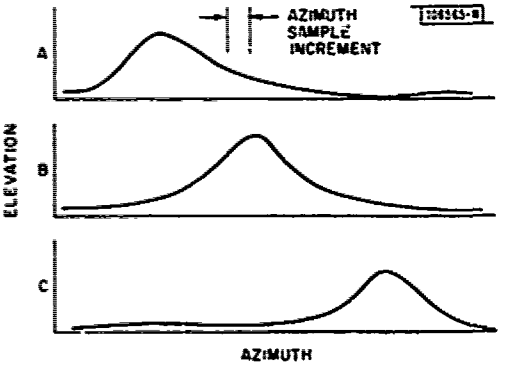


Fig. III-4. Schematic representation of situation in which a higher sampling rate in azimuth than in elevation could cause unmodified one-of-eight peak-picking algorithm to produce false peaks.

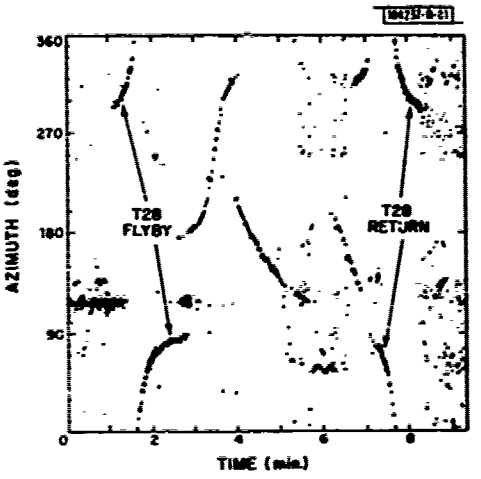


Fig. III-5. Speckle plot for 9-min. interval containing T28 flyby and return flyby. Azimuths of detected noise sources shown as a function of time.

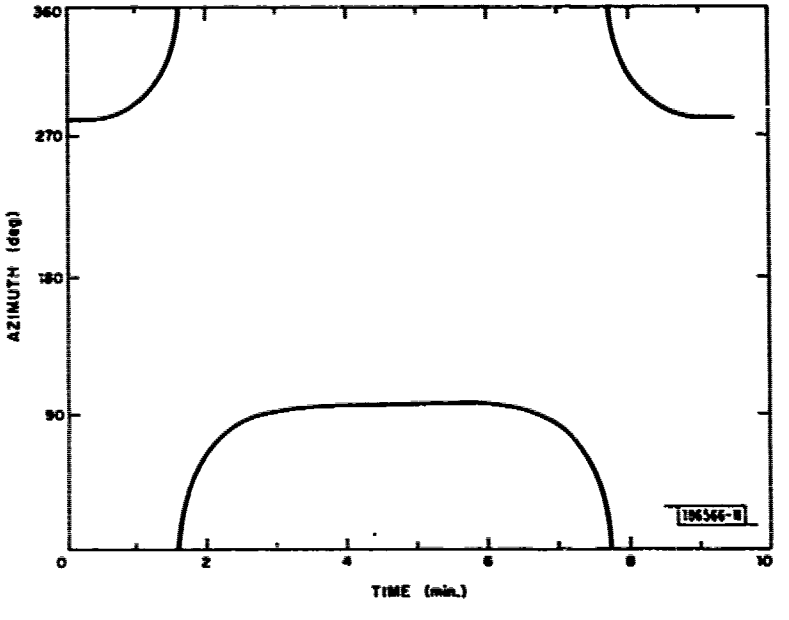


Fig. III-6. Theoretical acoustic azimuth vs time for T28 flyby and return flyby of Fig. III-5.

There would be no need to modify the one-of-eight algorithm if azimuth and elevation increments were about equal and both were sufficient for sampling the power function in azimuth and elevation. But to always sample at an equal and high-enough rate in both dimensions to handle the worst case would be computationally very costly. We have modified the algorithm to allow us to sample more coarsely in elevation than in azimuth, with a corresponding reduction in computation load. We typically sample azimuth uniformly every 3°, but use no more than eight elevation values.

Some 60 to 90 peaks of response are typically found per analysis interval, resulting from analysis at eight frequencies and eight elevations. Each of these peaks is characterized by azimuth, elevation, frequency, and sound-pressure level. If the azimuth values for each of these peaks are plotted vs time, the result is a speckle plot of the type shown in Fig. III-5 for a T28 aircraft. All peaks with an amplitude greater than −10 dB relative to the highest peak during each analysis interval are plotted using a symbol whose density is a function of the relative sound-pressure level in the analysis interval. From this figure it is possible to discern several aircraft tracks and, in particular, that of the T28 whose theoretical acoustic azimuth vs time is shown in Fig. III-6. The T28 flew on a straight-line course and, at a range of several kilometers, reversed course and flew on a straight line in the reverse direction.

Examination of the speckle plot reveals that while the azimuth tracks are clearly discernible to the eye, there is enough clutter to make the straightforward application of tracking algorithms difficult. The next steps in our present processing sequence are to cluster and prune the peaks.

Each source of sound can emit multiple frequencies, and we would like to associate these peaks together in each analysis interval. We have done this using standard clustering techniques.[2] Figure III-7 schematically shows peak location plotted as a function of azimuth and

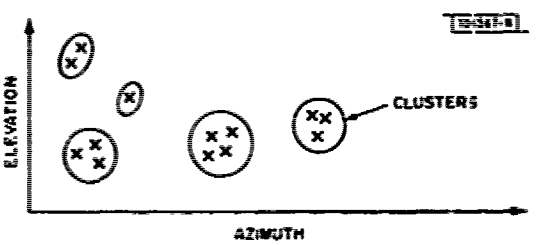


Fig. III-7. Schematic of peak clustering in azimuth-elevation space. Nearby points are clustered and replaced with an appropriately weighted average which is also assigned a significance rating.

elevation, and also shows clusters formed from peaks that are close together. In forming clusters, a distancing function is used which is the weighted sum of the absolute difference in azimuth plus the absolute difference in elevation, weighted to give more emphasis to azimuth differences than to those in elevation. The algorithm assigns each peak a weight equal to its sound-pressure level multiplied by its probability of being nonrandom. The required probability is determined from average power levels and the assumption that noise power has a Rayleigh distribution. It then finds the closest two peaks according to the distancing function and combines these into one cluster. The resultant cluster is located at a weighted average azimuth and elevation and is assigned a weight equal to the sum of its parts. The cluster is now used instead of its constituent peaks, and the algorithm is repeated until no peaks or clusters can be found that are within some specified maximum distance in azimuth and elevation. The resultant clusters are then assumed to represent targets.

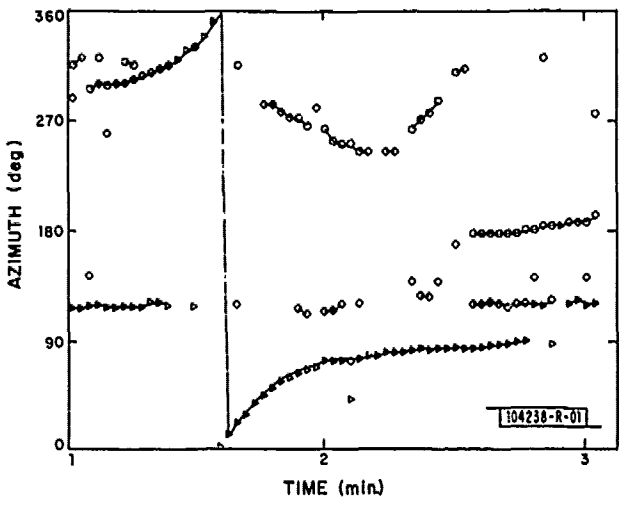


Fig. III-8. Results of single-site processing through clustering, pruning, and single-site azimuth tracking. Time scale same as Fig. III-5. Triangles indicate clusters judged most significant. Squares are next, and pentagons least significant. Points linked by lines were linked by azimuth tracker.

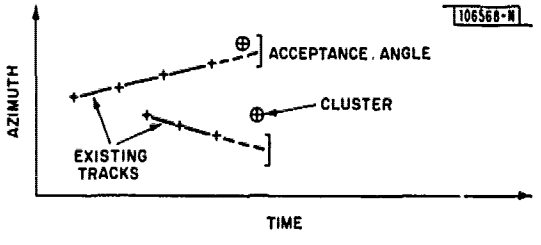


Fig. III-9. Schematic representation of assignment of new azimuth estimates to existing single-node azimuth tracks.

There are many low-level interfering sources of sound in the environment, and, for our initial processing, pruning was used at two levels. First, no peak was used as input to the clustering process which was more than 10 dB below the biggest peak in the analysis interval. Second, only the three clusters with the highest weights were used as input to the tracking filter. This considerably reduced the processing load and made it possible to use a simple association scheme for the tracker. The penalty paid for this pruning and the procedure used for selecting frequencies for spatial analysis were to limit the formation of tracks to a few dominant sound sources. Despite these limitations, it was possible to form clusters that clearly delineated aircraft tracks for distances up to 2 km from the nodes. Figure III-8 shows the clusters formed for a 2-min. segment of the flight of the T28 aircraft, together with the azimuth tracks formed from these. We expect that performance improvements will be achieved in a number of ways, including array processing before frequency selection and using target handover information to control processing.

A straightforward procedure has been used for azimuth tracking. The clusters formed in each analysis interval are compared with the extrapolated values of azimuth for existing tracks, as shown in Fig. III-9. If the cluster falls within some acceptance angle range of an existing track, then it is associated with that track, and used to update its tracking filter. If a point falls outside an existing track, it is treated as the initial point of a new track. In the current implementation, if a track is not updated it is terminated without coasting. Using an acceptance angle of 10°, the α-β tracking filter, which is presented in more detail in Sec. C below, produced the tracks shown in Fig. III-8. The data shown in the figure, plus similar results from a second array, were input to location algorithms described previously to produce the locations shown in Fig. III-1.

C. SINGLE-NODE AZIMUTH TRACKING FILTER

Azimuths assigned to clusters of peaks include random estimation errors. These errors are reduced by processing the clusters with a tracking filter. The tracking filter produces target azimuth estimates based on past data points and the current data point. It essentially puts a smooth curve through the data points.

The type of filter chosen for this is the α-β filter, which is a simplified Kalman filter that uses a first-order motion model. The formulation given here is based on the work of Bridgewater[3] and is that used to produce the tracking results shown elsewhere in this report.

Let Θ be the symbol representing azimuth, and T be the analysis data interval between the production of groups of clusters. Then, the prediction of the current value of Θ and its derivative from the prior estimates of the same quantities are

$$\Theta'_k = \hat{\Theta}_{k-1} + T\hat{\dot{\Theta}}_{k-1}$$

and

$$\dot{\Theta}'_k = \hat{\dot{\Theta}}_{k-1}$$

where the k subscript refers to time, and the symbol $\wedge$ identifies the previous estimates. These predictions, together with the azimuth $\rho_{k'}$ of a new cluster can be used to calculate updated estimates of $\hat{\Theta}$ and $\hat{\dot{\Theta}}$ as follows:

$$\hat{\theta}_k = \theta'_k + \alpha_k(\rho_k - \theta'_k)$$

$$\hat{\dot{\theta}}_k = \dot{\theta}'_k + \frac{\beta_k(\rho_k - \theta'_k)}{T} \quad .$$

These predictions are also used for data associations as explained in conjunction with Fig. III-9.

These estimates, which are the state of the filter, are the output of the tracking process. They are controlled by the parameters α and β which are computed as follows:

$$\alpha_k = \frac{D_{k-1}}{D_k}$$

$$\beta_k = \frac{\beta_{k-1} + \delta_{k-1} + \phi}{D_k}$$

where

$$D_k = 1 + \alpha_{k-1} + 2\beta_{k-1} + \delta_{k-1} + \phi$$

and

$$\delta_k = \delta_{k-1} + \phi - \beta_k^2 D_k$$

where ϕ is a controlling parameter that prevents α and β from going to zero as time $\rightarrow \infty$. It is given by

$$\phi = \frac{qT^2}{r}$$

where q is the covariance of the model error, and r is the covariance of the measurement error.

In initial tests, q and r have been assumed constant and, hence, so has ϕ. The value of ϕ was chosen to give a compromise between smoothing the azimuth track and losing track. If ϕ is made too large, then the tracking filter output is predominated by new data points. As a result, there is not much smoothing. If ϕ is made too small, new data points have little effect and the estimated curve follows a straight line, with a consequent loss of track. A value of 0.1 was found to be a good compromise for the cases studied.

The initial conditions used for initiating the α-β filters were described in Ref. 4. That is, given two initial measurements ρ_1 and ρ_2, we initiate a filter with

$$\hat{\theta}_2 = \rho_2$$

$$\hat{\dot{\theta}}_2 = \frac{(\rho_2 - \rho_1)}{T}$$

and parameters $\alpha_2 = 1$, $\beta_2 = 1$, $D_2 = 6$, and $\delta_2 = 2$.

D. ACOUSTICAL DATA ANALYSIS PROGRAM (ADAP) ENHANCEMENTS

Enhancements have been made to both the analytical and plotting capabilities of the ADAP program which is used for algorithm development and testing. In addition, changes have been made to the structure of the program to convert it to a multiple overlay program. This, in turn, increases the memory space available for analysis, which had proved to be a limitation.

The major analytic extension to ADAP is the generation of peaks directly from any of the standard input-data formats. Peaks are located in azimuth and elevation, using the extended one-of-eight algorithm, and are written into "peak" files. These files have data records with the format:

Field 0: Time
Field 1: Frequency
Field 2: Elevation
Field 3: Azimuth
Field 4: Sound Pressure Level
Field 5: Signal-to-Noise Ratio

Other analytic extensions include:

(1) Ability to individually specify channel gains.

(2) Ability to use any selected set of input channels for analysis.

(3) Ability to specify offset time into input data at which to start a run, and also the duration of the data to analyze.

(4) Choice of methods for choosing frequencies at which to do analyses.

Extensive additions have been made to the plotting capabilities of ADAP. The first extension is to allow multiple time series, spectra, or power-vs-azimuth curves to be plotted on a single graph. A typical example for time series is shown in Fig. III-10. The user can select the number of graphs that appear on a single page, and the program will automatically produce as many output pages as are necessary to provide the requested plots.

Fig. III-10. Typical ADAP graphical output showing multiple time series.

In the case of time series, the user specifies the channels for which plots are desired. The starting time and the number of seconds of data to plot are also specified. The scale on all traces is made the same so that they can be directly compared. This scale is automatically chosen unless the user wishes to override the default and specify the maximum sound-pressure level. The parameters of the plot are indicated on the bottom of the page along with an 80-character comment field which the user can input prior to running the analysis which creates the plotting file.

Spectral plots are similar to waveform plots, except that the user specifies starting time and the number of seconds over which to average the spectra of individual channels. Also, the user can specify the minimum and maximum frequencies to be plotted.

Power-vs-azimuth plots are parameterized by both frequency and elevation at which the power-vs-azimuth computation was performed. Again, the user can specify how many plots appear on a page and whether the scale is automatic or specified. The user specifies the time offset into the file, and the data are plotted for the next data interval. By specifying ranges for frequency and elevation, the user can limit the number of graphs plotted.

An alternate to the power-vs-azimuth plots is also available in the form of a wavenumber plot, as shown in Fig. III-2. This is a contour plot of power vs wavenumber at a single user-specified frequency. The wavenumbers have both north-south and east-west components, with zero wavenumber value lying at the center of the circle. Target bearings are measured by clockwise rotations from north around the circle. Target elevations appear as a radial distance, with targets overhead appearing at the center of the circle and those on the horizon appearing on the outer circle. Bearing marks are given every 10° on the outer horizon circle. The inner circle indicates the maximum specified elevation – in the case of Fig. III-2 it is 72°.

The power contours are interpolated from the power vs azimuth and elevation data, and are displayed with a user-specified decibel increment between contours. Different contour line patterns are used to indicate uphill and downhill directions. The highest-level contours are solid, followed by small dots and so forth. The user specifies the time and frequency of interest, and can specify the maximum scale level or have it automatically selected. The minimum level is always automatically selected.

The speckle plot output, shown in Fig. III-5, was selected as the ADAP graphical form to use for peaks. Peaks are plotted by azimuth and time, using a symbol to indicate the strength of the peak. Because of the limited range of symbol densities available, the symbol chosen was based on a normalized value relative to the highest peak in the analysis interval. The highest peaks received an "*" and those 10 dB down from the peak are indicated by a ".", with all peaks below this level not being plotted. The result is a plot showing the relative strength of target peaks aggregated over all frequencies. While having some limitations due to symbol density range, it does give a good indication as to detectable target azimuths and has proved invaluable in interpreting experimental data.

The ADAP now includes 15,000 lines of code. We could not fit it into the 64K bytes of data and 64K bytes of instruction space permitted for a single task on our PDP-11/70 computer and still have the features we desired. Therefore, during the past several months ADAP was restructured as a multi-program set running under the control of an executive, as shown in Fig. III-11. The executive module, ADAP, handles all communications with the user. It takes in commands either directly or from files, and stores the resultant parameters. When a command is given to run an analysis, the executive writes all the information into a temporary file and passes control to the ARUN program which performs the analyses.

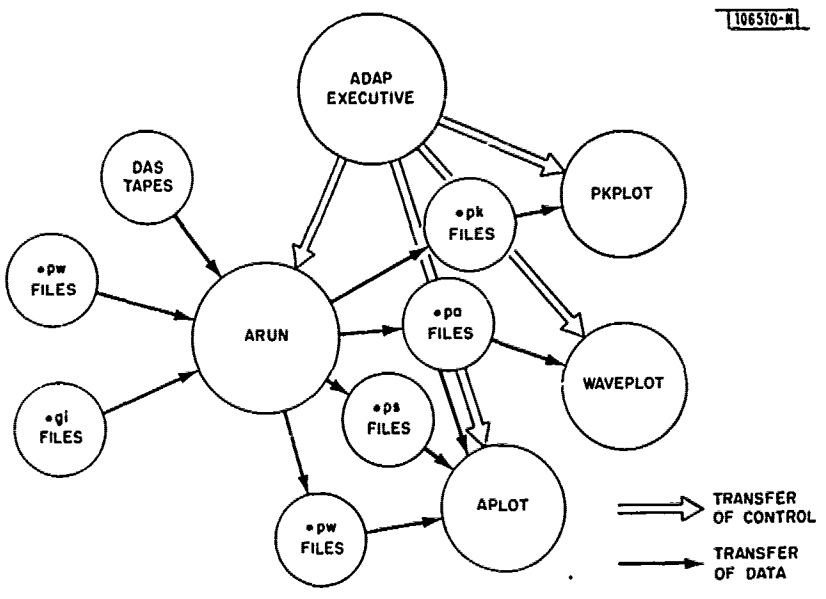


Fig. III-11. Interrelationships of ADAP programs and files. Data file contents are determined by a .xx extension to file name. Those shown are: (.pk) power peaks, (.pa) maps of power in frequency – elevation – azimuth space, (.ps) spectra, (.pw) waveforms, and (.gi) information about certain waveform files.

ARUN reads all the analysis parameters from the temporary file, including the input-data format and the output file to be created. It then performs the specified analysis and returns control to the ADAP executive program where the user can request to obtain plots. When this is done, ADAP again uses a temporary file to transfer control parameters to the APLOT program which performs the plotting of time series, spectra, and power-vs-azimuth graphs. When wavenumber or speckle plots are requested, then the WAVEPLOT and PKPLOT programs are run respectively.

The two major advantages of this organization are the ease of expansion to ADAP, and the extra memory space made available for analyses. Memory space for all the arrays and matrices needed in the analysis of experimental data is still a limitation. However, we are now able to run with blocklengths and numbers of frequencies compatible with those to be used for real-time processing in the array processor. Overcoming the remaining memory space limitations will be one of the major ADAP issues to be addressed in the upcoming year.

REFERENCES

1. Semiannual Technical Summary, Distributed Sensor Networks, Lincoln Laboratory, M.I.T. (31 March 1980), DTIC AD-A091766.
2. R. Gnanadeskian, Methods for Statistical Data Analysis of Multivariate Observations, Chapter 4 (Wiley, New York, 1977).
3. A. W. Bridgewater, "Analysis of Second and Third Order Steady State Tracking Filters," in AGARD Conf. Proc. No. 252, "Strategies for Automatic Track Initiation," Monterey, California, 16-17 October 1978, pp. 9-1 to 9-11.
4. R. A. Singer and K. W. Behnke, "Real-Time Tracking Filter Evaluation and Selection for Tactical Applications," IEEE Trans. Aerospace Electron. Systems AES-7, 100-110 (1971).

IV. ADVANCED NODE ARCHITECTURE

The test-bed hardware and software described in Sec. II of this SATS have been designed to provide the capability needed for initial DSN experimentation. The hardware is primarily sized to provide for data acquisition, real-time signal processing, and basic communication capabilities. We have started to investigate a new node computer architecture which can augment the capabilities of the test-bed nodes and can serve as the basis for future development of small low-power nodes. The architecture being investigated is a multiple-microprocessor node, with processors interconnected by high-speed packet-oriented busses.

One of the modules of this processor will interface to a radio unit which can measure ranges between radio units located at different nodes and can also provide DSN internode communication. This module will support communication management in the DSN and will provide a self-location capability by means of the node-to-node range measurements. Other processor modules will augment signal-processing capability, provide processing power and memory needed to support investigation of advanced DSN concepts in real time in the test bed, and support input and output between the node and other attached peripheral devices.

The following sections summarize progress with the real-time network system software for the advanced node architecture (and for the DSN test bed in general) and present an overview of one strawman hardware design for the advanced node. Earlier work on the basic system software (REal-time NEtwork kernel, RENE), which is being developed to run on the existing DEC PDP-11 machines in the test bed as well as the new advanced node microprocessors, was presented in the previous SATS.[1] The hardware design summarized here represents a strawman which is being used to refine ideas and help identify critical problem areas.

A. REAL-TIME NETWORK KERNEL (RENE)

Design of RENE has progressed, with implementation on the operating system independent part (OSD) proceeding as reported in Sec. II-B of this SATS. As discussed in Sec. II-B, OSD is a major part of the RENE code and, in particular, contains memory pool managers that form a substantial part of the interface between an application process and the RENE network communications processes.

Two versions of a memory mapper design have been developed, but we are still not satisfied with the part of the designs that provides for swappable memory to be shared among real-time processes. In addition, we need to review the designs with respect to commercially available multi-microprocessor systems with hardwired or minimally controllable memory maps.

The process scheduler in DAK has been studied with a view to enhancing it in two ways for use in RENE: first, making it suitable for implementation under UNIX as well as on stand-alone computers, so that multi-process programs can be debugged under UNIX; and second, adding a time-protection system, protecting the allocation of real time just as a memory-protection system protects memory, so that processes can be slowed down when they attempt to overdrive the computer. Time-protection systems can be of considerable value and we are now reviewing a general approach given in Ref. 2 to see if it can be adapted for use in the DSN.

Detailed work is well along on the communication subsystem design reviewed in Sec. IV of Ref. 1, with particular emphasis on the user interface and on the interface to an underlying communications network. In the remaining part of this section, we examine the communication issues of network congestion and the use of datagrams as the communication interface for RENE.

Along with reliability, throughput, and response time, congestion is a major problem facing the designer of a distributed system. We reported on reliability in Ref. 1 and proposed that in RENE reliable communication would consist of requests made from user processes to server processes, each request would have a corresponding response, and user processes would be responsible for retries. This meant that user processes had to keep a copy of the original request, use timeouts to force retries, and be prepared to retry upon receiving negative acknowledgment messages from the network or server.

Note that many DSN application programs will not use reliable communication, but will use nonclassical algorithms involving unreliable broadcast communication between nodes. From the point of view of the operating system, such communication is simple to implement. Here, we focus upon the more difficult problems related to reliable communication within and between nodes.

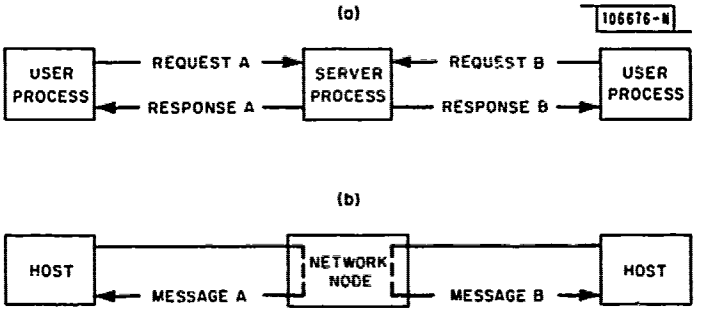


Fig. IV-1. Similarity of communication network congestion and server congestion in a distributed system. (a) Server in a distributed system; (b) node in a communication network.

Distributed operating systems suffer from congestion that arises in the same way congestion arises in a network. Consider Fig. IV-1(a) in which we depict a distributed operating system with one server, two users, and two request messages and their associated response messages. In Fig. IV-1(b) we redepict these same objects after renaming them: each request message has been identified with its associated response message, as if the two were the same message, while the server has been renamed a "network node" and the users have been renamed "hosts." As this renaming suggests, a distributed operating system will behave with respect to congestion like a network, with servers behaving like network nodes and susceptible to congestion in the same way.

When a distributed operating system is built on top of a real message-passing network, it is important to try to decouple the congestion problems of the operating system from those of the underlying network. If this is not done, congestion in a network path can cause congestion in a server and vice versa. One good means of decoupling the congestion seems to be what we call the perfect-sink/memoryless-link rule. The perfect-sink rule says that the operating system will without delay accept any message presented by the underlying network, thus behaving like a perfect sink for messages. The memoryless-link rule says that links provided by the underlying network have no memory and, like wires, cannot save messages arriving at their ends. These two rules are, of course, the same rule viewed from opposite sides of the

network/operating system interface. RENE will be made to approximate a perfect sink as nearly as possible.

Now consider the servers. They can take at least two different approaches to congestion control. A magnetic-tape driver, for example, prohibits a sender from transmitting the data before a buffer is ready for it. This is positive congestion control: a request for receiver buffer space must be positively acknowledged before data can be sent. But a disk-file manager with many users may be best off by allowing oversubscription of its receiving buffer space. If this is done, then some data-write requests would have to be negatively acknowledged with a try-again-in-a-moment message. This is negative congestion control. Both techniques will be required, depending upon the specific driver and circumstances.

The impact of our reliability and congestion attitudes is to make the interface between RENE and its underlying communication networks consist of datagrams which are messages presented by RENE for transmission through the network and which have the following properties: there is in general no advanced preparation for a particular message by either the network, or as far as the network knows, by the receiver of the message; and the network does not have responsibility for infallibly delivering the message.

Table IV-1 lists the information required in a datagram header by the current RENE design. Five of the items are variable size, and in each case the size has been allowed to vary in units of 2 bytes from 0 to 30 bytes, so that the size may be encoded into 4 bits. The general meaning and use of most of the header contents should be clear from the table. However, following are additional comments and explanation related to "circuit" and "flags."

Throughput and response-time considerations suggest that the physical network be partitioned into portions we call circuits which have specified response properties. For example, a high-performance circuit might be used to exchange datagrams between processors in a single DSN node; another circuit might provide a guaranteed transfer rate to neighboring nodes while providing a specified maximum delay and probability of error. Many different circuits and several different kinds of circuits are expected to be required. Several datagrams with a common circuit may sometimes be combined to reduce the overhead of datagram headers and improve throughput.

Flags in the datagram header are in anticipation that the underlying communication network may provide information about some of its congestion control activities. In particular, it might give information about datagrams deleted by the communication network for congestion control purposes. However, it is not assumed that the network must supply such a service or that the service, if provided, is perfect.

In the current RENE design, connections between nodes are opened with the help of a directory server which provides the source-node address, destination-node address, circuit IDs, checksum format, and the maximum number of information bytes allowed in a datagram using the circuits. The rest of RENE and application code need not be able to interpret the source and destination addresses or circuit IDs in terms of network implementation or configuration. The input to the directory server from a typical RENE program trying to open a connection is a character string pathname plus some access-type options (e.g., read, write, execute). Code for creating circuits is embedded in directory servers.

B. STRAWMAN HARDWARE DESIGN SUMMARY

Here we summarize a strawman design, designated Distributed Processing Unit 1981 or DPU81, for a packet bus interconnected multi-microcomputer system which might be the basic

TABLE IV-1

DATAGRAM HEADER CONTENTS

Name	Bytes	Use
size	4.5	Number of information bytes in datagram (from 0 through 32,768), and sizes of source, destination, circuit, ID, and sum fields.
source	0 to 30	Address of network node that is datagram source.
destination	0 to 30	Address of network node that is datagram destination.
circuit	0 to 30	Identifier of network circuit whose resources the datagram is to use.
ID	0 to 30	Unique datagram identifier. Unique for datagrams with the same time-stamp, source node, and destination node, provided that the lifetime of any datagram ID in the network is <30 min.
sum	0 to 30	Datagram checksum or encrypted signature that verifies that datagram has not been changed since originally created. This signature is verifiable by the general public, including each network node through which the datagram passes.
time	3.0	Time datagram was created, in units of 1/1024 s, with 4-h ambiguity.
flags	0.5	The following flags:
		DG_CONTROL — When true indicates that the information bytes contain a control message from the network to a source or destination node about the datagram identified by the time-stamp, source, destination, and ID.
		DG_DESTINATION — On if control messages about the datagram should be sent to destination instead of sent back to source.

computing unit in future DSN test-bed nodes. All DSN functions including self-location, communication, resource management, tracking, and signal processing are to be supported. The architecture should yield a moderately priced node usable for research in distributed systems, and also be a model for the development of future low-cost nodes. The architecture should not be overly committed to any particular sensor, algorithm, or microprocessor chip. The design should be such that we can make effective use of existing DSN processing elements, such as the PDP-11/34's and FPS-120B signal processors now in the DSN node. The DPU81 architecture includes an integral signal processor, but we do not plan to develop such now and expect to use the FPS-120B's indefinitely. The system design should incorporate features which reduce the difficulty of construction, hardware debugging, and maintenance.

The design given here is intended to offer a superset of the features that will actually be implemented. The final selection of features to include will be made while finalizing the architecture and proceeding to detailed design and development during 1981.

The DPU81 design given here is not the only way to obtain a multi-microcomputer node with the same general architecture. It is possible to build DPU81, or something like it, around existing commercial microcomputer printed-circuit boards. This might well increase cost and power consumption as well as require compromising on some desired features. But it could reduce development effort. The desirability and utility of that approach depends greatly upon the fast-changing commercial microcomputer market. Before further detailed work on the existing design, one of our next tasks will be to review commercial offerings, which have progressed considerably in the last few months, and evaluate if we can take advantage of new developments.

In the subsections below, we summarize existing DPU81 design documentation with emphasis on how the microcomputers are related to each other.

1. Physical Layout

The DPU81 design calls for a 19-in. rack-mounted unit about $10\frac{1}{2}$ in. high capable of mounting up to 32 printed-circuit boards each holding at most 150 ICs. The backplane contains a 64-line packet communication bus which connects all boards. Each board nominally consists of a 50 IC communication bus interface plus a 100 IC microcomputer. The microcomputer will typically include 256K bytes of RAM and either a Motorola MC68000 general-purpose microprocessor IC or an 8X300 I/O controller IC. The design goal is an average of 0.25 W regulated power per IC, or with a power regulator operating at 75-percent efficiency, 50 W unregulated power per board.

Since an MC68000 has the approximate processing power of a PDP-11/44, though with an instruction set whose high-level language capability is closer to that of a VAX, the maximum-sized DPU81 has a very large amount of computing power. By using smaller numbers of boards, the DPU81 may be scaled down to a lower limit of two boards: one I/O processor and one general-purpose processor which would have a combined power consumption of 100 W. A configuration which would be more appropriate for a complete DSN node might consist of 6 processor boards and 5 signal-processing boards, and would have 6 times the general-purpose processing power of our current PDP-11/34, twice the signal-processing power of our current FPS-120B, and hopefully draw only 600 W of unregulated power.

It is desirable to be able to extend the microcomputers beyond 150 ICs, so the backplane makes provision for 64-line short busses, called daisy chained busses, connecting several

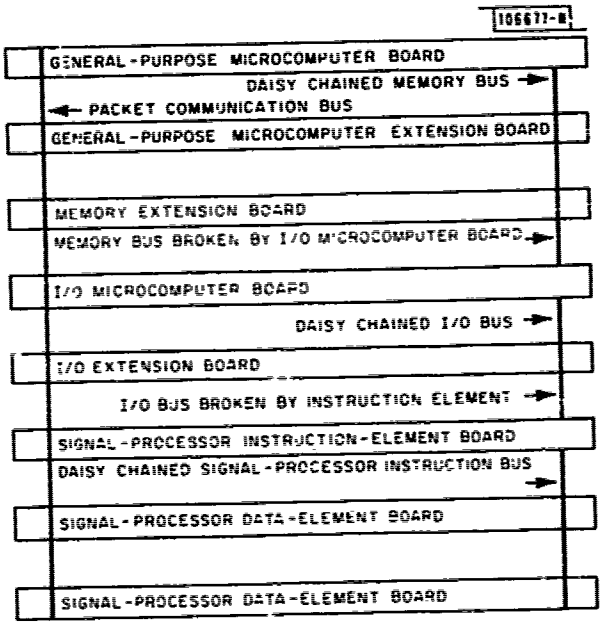


Fig. IV-2. Packet communication and various daisy chained busses.

adjacent boards. Each microprocessor board is the head of a daisy chained bus, which connects this board with a set of microcomputer extension boards to its right (see Fig. IV-2). The MC68000 board is the head of one such 64-line daisy chained bus called the memory bus, which can accommodate a processor extension card with a memory map, cache, and an additional 256K bytes of RAM, and a memory extension card with 1M byte of memory. Similarly, the I/O processor board is the head of a daisy chained bus upon which I/O interfaces can be constructed, though actually the principal idea behind using the 8X300 I/O controller is that it is so fast in I/O control applications that most external devices can be interfaced to it by cables, without any special-purpose hardware logic. I/O interfaces can also be constructed for the MC68000 memory bus, but that would disrupt some of the reliability features outlined below.

Single-processor boards are also accommodated on the bus. The DPU81 design specifies that there be two types of signal-processor boards: an instruction element and a data element. Each instruction element is the head of a daisy chained signal-processor instruction bus containing one or more data elements. Thus, it is possible to build Single Instruction Stream Multiple Data Stream (SIMD) signal processors with varying numbers of data elements from these two boards. A review of DSN signal-processing requirements for various sensors – acoustic, infrared, radar – indicates that a SIMD design is appropriate. In the strawman design the instruction elements are very simple, and it is intended that a general-purpose MC68000 microcomputer will supervise the signal-processor boards and provide bulk memory for signal processing, with the packet communication bus being used for communication between this genera'-purpose processor and the signal processors.

2. Data Sizes and Rates

The DPU81 design assumes that signal pro .ssing will use 32-bit data and 64-bit parameters. In the immediate future these are assumed to be fixed point, because the MC68000 does

not yet support floating point. The design assumes that the signal processors in one node might eventually be required to do up to 32 million real arithmetic operations per second, and require packet bus communications at a rate of 4 million data words per second to sustain this rate. For example, a SIMD signal processor with 1 instruction element and 8 data elements should be able to reach the maximum arithmetic rate if the algorithms partition properly, as they do for DSN signal processing. Given the above assumptions, the packet bus throughput required for signal processing is 16M bytes/s (4 bytes per data word). To provide for this, allow for unanticipated growth, and provide for other interactions on the bus, we have specified a bus rate of 32M bytes/s. This should be easily achievable using 32 data lines and an 8-MHz clock rate. Note that this high speed is driven primarily by signal-processing activities. An option to be investigated in the near future is the separation of the signal-processing traffic from more general interprocessor communication.

3. Reliability and Clocking

For reliability and maintenance reasons, it is very desirable to be able to run two identical boards synchronously through the same states comparing their outputs and even internal signals. There are two important applications of this. One is repairing faults during manufacture and maintenance, and the other is isolating faults to board level under remote control in an operational system. With this in mind, the DPU81 design provides for this synchronous operation capability.

The synchronous operation of two boards will allow the use of arbitrary application software as a diagnostic program and will isolate faults to the board level under remote control. In general, only one of two boards running synchronously will be enabled to output packets or other signals. For the case of packet transmission, only one computer outputs the data part of the packet, but each computer outputs a separate checksum, so that receivers can check whether the synchronized microcomputers agree on their result. Identical multi-board as well as single-board microcomputers can be synchronized.

Given synchronized boards running application software, a technician using special probes can determine points on the board where signals disagree, and can search for the earliest point of disagreement, thereby isolating a fault. By synchronizing microcomputers under remote control, faults can be isolated to microcomputer level in an operational system.

It is more difficult to synchronize I/O microcomputers. First, most deployed DSN nodes will not contain two I/O microcomputers with identical external attachments. Furthermore, special cross-connections would be required to synchronize two I/O microcomputers because the external data entering them are not synchronized with the system clock. The problem of remote fault diagnosis for the case of I/O processors is an area requiring more study to determine the best approach.

The ability to synchronize boards requires special attention to the problem of reliability of the system master clock, and to accommodating microprocessors with different speeds. The system master clock is made reliable by providing redundant master clocks, one on each I/O microcomputer board. These boards negotiate at system startup time as to which shall become the master clock. The I/O boards can disconnect themselves from the packet communication bus, can run independently and asynchronously from that bus, and can monitor that bus. These boards also contain a small nonvolatile memory to remember the past successes and failures

of boards or the bus. To accommodate various-speed microprocessors, synchronized 8-, 12-, and 16-MHz master clocks are provided.

4. The Packet Communication Bus

The packet communication bus provides all inter-microcomputer communication. The data rate of the bus was set at 32M bytes/s above. Allowing each microcomputer on the bus to transmit an average of 1000 packets per second, with the designed maximum of 32 microcomputers on one bus, gives a maximum aggregate bus rate of 32,000 packets per second.

A design objective for the packet communication bus is that its interface be implementable with a modest amount of hardware. A nominal parts count of 50 ICs is the goal. For this reason, packet buffer memories have been eliminated from the interface, and packets are to be transmitted directly from the main memory of one microcomputer to the main memory of another. The speed of large main memories is limited by existing technology to rates of about one 32-bit word per 250-ns average (for burst transmissions at sequential locations using a memory IC feature called page mode), or 16M bytes/s. This is half the desired bus rate. Therefore, the DPU81 design has two separate 16-bit data busses for transmitting two independent packets at once, each at a 16M byte rate. Thus, there are four microcomputers involved at one time in transmitting two packets and receiving two packets. This scheme requires that the microcomputer memories be completely co-opted by the packet bus interface when they are transmitting or receiving packets. This is a minor problem for I/O processors which must have faster, hence smaller, main memories. We may possibly decide that it is better to have 3 or 4 packet busses each with 8-bit data paths and 8M byte/s rates, rather than just 2 busses.

A block diagram of a communication bus interface is shown in Fig. IV-3. The bus actually consists of three separate busses: a signal bus with 8-bit data, and two packet busses with 16-bit data. The signal bus is scheduled using a priority algorithm with 64 priority levels. Special signals called packet-enable signals are used to schedule the packet busses. Packet bus scheduling is overlapped with transmission of previous packets. Once a packet transmission is ready to start, the transmission processors in the interface take over and transmit the packet.

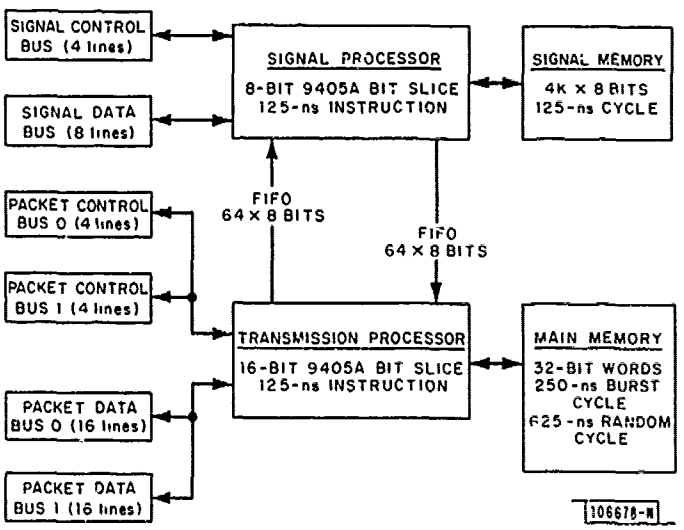


Fig. IV-3. Packet communication bus interface.

The general-purpose and I/O processors control the interface by placing programs in main memory (not part of the interface) which are executed by the transmission processor. There are actually 256 such programs, which are called channel programs, corresponding to 256 separate processes, called channels. One of these is used to send signals to other microcomputers, and executes send-signal instructions. Another is activated every time a signal is received and executes store-received-signal instructions. Other channels are activated whenever a packet-enable signal is sent or received designating a channel which is to be executed to send or receive a packet. These channels execute programs that send or receive blocks of main memory that comprise the packet. Packets may be discontinuous in main memory. Other channel instructions may be used to store packet-enable signals in main memory and interrupt other processors on the main memory bus, such as the MC68000 and 8X300 processors.

Signals may also be broadcast to all processors or to one or more groups of processors. Broadcast of signals is used to start up the bus: the bus controller interrogates the microcomputers on the bus to see what their serial numbers and types are, and assigns virtual addresses to the microcomputers. By assigning the same virtual address to two or more microcomputers, these can be rigged to run synchronously through the same states. Packets may also be broadcast to more than one microcomputer, but in this case the hardware takes no responsibility for ensuring that the receiving microcomputers are not busy with other packets. If they are, they will be unable to process the packet-enable signal which they receive.

REFERENCES

1. Semiannual Technical Summary, Distributed Sensor Networks, Lincoln Laboratory, M.I.T. (31 March 1980), DTIC AD-A091766.
2. R. L. Walton, "CBL Report on the Software and Hardware Problems of University Computer Centers Serving Fast-Real-Time Users, 1969-1973," Harvard University Psychology Department (1975).

V. MISCELLANEOUS ACTIVITIES

A number of project accomplishments and activities which do not fit directly into a specific larger category are very briefly summarized here.

We have continued to develop our working relationship with other research groups, including students and members of the M. I. T. faculty who are interested in distributed problem solving, surveillance, and information distribution. One Masters Thesis† has been written, and we expect additional contributions from this relationship in the future. A number of meetings have been held with a group involving Lincoln, M. I. T. faculty, and students interested in distributed-decision-making problems. The group includes a cross section of people with computer science, artificial intelligence, system theory, estimation and control theory interests. As a result of these meetings we have prepared an informal memo describing a somewhat simplified DSN surveillance and tracking system problem which is based upon the use of idealized azimuth-only sensors. We have also drafted another memo which gives a more detailed model for acoustic azimuth sensors and for two-dimensional radars. Prof. V. Lessor of the University of Massachusetts, Amherst, who is also interested in distributed-decision-making problems, participated in one of the meetings at M. I. T. and we are continuing discussions of issues with him.

Small amounts of acoustic data, some raw and some processed, have been given to Prof. Jae S. Lim of M. I. T. and to DSN researchers at Stanford and Carnegie-Mellon University. Prof. Lim is investigating two-dimensional Maximum Entropy spectral analysis. The data sent to Stanford will be used to check out their transfer of our ADAP software from PDP-11 computers to a VAX, and will also be available to them as input for their own algorithms. A member of the Stanford research team visited with us for three days to gain an understanding of our signal-processing algorithms and requirements and to obtain information related to the transfer of ADAP and the use of the data.

We have reviewed ongoing work at the University of Southern California Information Sciences Institute in anticipation of developing a DSN self-location capability based upon measurements of ranges between nodes. This involved reviewing available documents, discussions with the principals, and some familiarization with the software they have actually developed. We concluded that their constructive approach to self-registration is a good point at which to start to develop operational algorithms. Among the important issues which now must be considered are distributed implementation and protocols for that implementation, handling of errors including the possibility of very large systematic errors resulting from multipath, how to handle absolute as well as relative positions, and how to handle at least some moving units.

During this reporting period, it became necessary to physically move the office and laboratory space of the DSN program within Lincoln Laboratory in order to provide adequate laboratory and office space for the project. The new space is several hundred feet away from the PDP-11/70 computer which provides most of the general computer support for the project. This required the installation of cable and short-haul modems between the computer and the new work areas in addition to other routine preparation of the areas.

---

†R. P. Hughes, "A Distributed Multiobject Tracking Algorithm for Passive Sensor Networks," Masters Thesis, Department of Electrical Engineering and Computer Science, M. I. T. (June 1980).

Also, as has been planned for some time, the software system for the PDP-11/70 was converted from Version 6 to Version 7 UNIX during this time period. This conversion was required in order to make use of new and improved UNIX features and user-level tools as well as to remain current with the mainstream of the UNIX community. The conversion included providing for ARPAnet service under Version 7 and, to our knowledge, we were the first Version 7 UNIX connected to the network. Other tools continue to bc converted and debugged in the new environment as the need arises or bugs are detected.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: ESD-TR-80-244	2. GOVT ACCESSION NO.: AD-A103045	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)*: Distributed Sensor Networks.	5. TYPE OF REPORT & PERIOD COVERED: Semiannual Technical Summary rept. 1 April – 30 September 1980
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): Richard T. Lacoss	8. CONTRACT OR GRANT NUMBER(s): F19628-80-C-0002, W ARPA Order-3345
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Lincoln Laboratory, M.I.T., P.O. Box 73, Lexington, MA 02173	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: ARPA Order 3345, Program Element Nos. 61101E and 62708E, Project Nos. 0D30 and 0T10
11. CONTROLLING OFFICE NAME AND ADDRESS: Defense Advanced Research Projects Agency, 1400 Wilson Boulevard, Arlington, VA 22209	12. REPORT DATE: 30 September 1980
	13. NUMBER OF PAGES: 48
14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)*: Electronic Systems Division, Hanscom AFB, Bedford, MA 01731	15. SECURITY CLASS. *(of this report)*: Unclassified
	15a. DECLASSIFICATION DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

None

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

multiple-sensor surveillance system
multisite detection
target surveillance and tracking
2-dimensional search-space cell
acoustic, seismic, radar sensors
low-flying aircraft
acoustic array processing
high-resolution search-algorithms

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

This Semiannual Technical Summary reports work in the Distributed Sensor Networks program for the period 1 April through 30 September 1980. Progress related to development and deployment of test-bed hardware and software, including deployment of three test-bed nodes, is described. A complete algorithm chain from raw data to aircraft locations, employing two acoustic arrays, has been developed and demonstrated experimentally using data collected from test-bed nodes. A straw-man design for a new multiple microprocessor test-bed node computer is presented. Also described is progress in the design and development of a real-time network kernel for the DSN test bed in general, and the new processor in particular.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

207650